

TD 2

Scrutation et Interruption externe

On considère un montage où le LPC2378 est connecté avec 8 LEDs sur la partie basse du port P4 (P4.0 à P4.7), 8 interrupteurs sur le deuxième octet du port P4 (P4.8 à P4.15) ainsi qu'un bouton poussoir sur le pin P2.13 du port P2.
Ecrire la fonction `init_GPIO()` qui permet de configurer les registres de sélection et de direction du LPC2378 pour respecter le montage considéré ci-dessus.

Quel est le registre qui va permettre de faire que le port P4 est de type GPIO ?

De manière générale la patte Px.y (port X, bit y), son contrôle est défini par un registre PINSEL n

comment déterminer n : c'est PINSEL $2x$ ou PINSEL $2x+1$

chaque bit de port est affectable à 4 périphériques différents : il faut donc 2 bits pour choisir 1 parmi 4 possibilités

conséquence, notre registre PINSEL n ne peut gérer que 16 pattes ...

PORTE : PINSEL0 pour les 16 premières pattes : P0.0 à P0.15 et PINSEL1 pour les 16 dernières de P0.16 à P0.31

PORT1 : PINSEL2 pour les 16 premières pattes : P1.0 à P1.15 et PINSEL3 pour les 16 dernières de P1.16 à P1.31

PORT2 : PINSEL4 pour les 16 premières pattes : P2.0 à P2.15 et PINSEL5 pour les 16 dernières de P2.16 à P2.31

PORT3 : PINSEL6 pour les 16 premières pattes : P3.0 à P3.15 et PINSEL7 pour les 16 dernières de P3.16 à P3.31

PORT4 : PINSEL8 pour les 16 premières pattes : P4.0 à P4.15 et PINSEL9 pour les 16 dernières de P4.16 à P4.31

pour les leds : PINSEL8, pour les inters : PINSEL8

pour le poussoir, P2.13 --> PINSEL4

Quels bits sont concernés : 2 bits de configuration pour 1 patte ==> on prend la paire de bit $2y$ et $2y+1$
si on tombe sur des nombres supérieurs à 32 on enlève 32 c'est un indicateur qu'on doit être sur un pinsel impair

Table 117. Pin function select register 8 (PINSEL8 - address 0xE002 C020) bit description (LPC2377/78 and LPC2388)

| PINSEL8 | Pin name | Function when 00 | Function when 01 | Function when 10 | Function when 11 | Reset value |
|---------|----------|------------------|------------------|------------------|------------------|-------------|
| 1:0 | P4.0 | GPIO Port 4.0 | A0 | Reserved | Reserved | 00 |
| 3:2 | P4.1 | GPIO Port 4.1 | A1 | Reserved | Reserved | 00 |
| 5:4 | P4.2 | GPIO Port 4.2 | A2 | Reserved | Reserved | 00 |
| 7:6 | P4.3 | GPIO Port 4.3 | A3 | Reserved | Reserved | 00 |
| 9:8 | P4.4 | GPIO Port 4.4 | A4 | Reserved | Reserved | 00 |
| 11:10 | P4.5 | GPIO Port 4.5 | A5 | Reserved | Reserved | 00 |
| 13:12 | P4.6 | GPIO Port 4.6 | A6 | Reserved | Reserved | 00 |
| 15:14 | P4.7 | GPIO Port 4.7 | A7 | Reserved | Reserved | 00 |
| 17:16 | P4.8 | GPIO Port 4.8 | A8 | Reserved | Reserved | 00 |

Table 117. Pin function select register 8 (PINSEL8 - address 0xE002 C020) bit description (LPC2377/78 and LPC2388)

| PINSEL8 | Pin name | Function when 00 | Function when 01 | Function when 10 | Function when 11 | Reset value |
|---------|----------|------------------|------------------|------------------|------------------|-------------|
| 19:18 | P4.9 | GPIO Port 4.9 | A9 | Reserved | Reserved | 00 |
| 21:20 | P4.10 | GPIO Port 4.10 | A10 | Reserved | Reserved | 00 |
| 23:22 | P4.11 | GPIO Port 4.11 | A11 | Reserved | Reserved | 00 |
| 25:24 | P4.12 | GPIO Port 4.12 | A12 | Reserved | Reserved | 00 |
| 27:26 | P4.13 | GPIO Port 4.13 | A13 | Reserved | Reserved | 00 |
| 29:28 | P4.14 | GPIO Port 4.14 | A14 | Reserved | Reserved | 00 |
| 31:30 | P4.15 | GPIO Port 4.15 | A15 | Reserved | Reserved | 00 |

PINSEL8=0; //configuration autoritaire , les 16 pattes sont configurées pour être des GPIOs

Pour configurer une seule patte : exemple de P2.13

on fait $2^2 = 4 \Rightarrow$ on travaille avec PINSEL4 ou 5

on fait $13 \cdot 2 = 26 \Rightarrow 26 < 32$ donc ce seront les bits 26 et 27 du premier registre PINSEL : PINSEL4.26 et PINSEL4.27

si on souhaite modifier seulement ces deux bits :

il faut faire un masque 0x03 <<26 permet de cibler les 2 bits

Table 113. Pin function select register 4 (PINSEL4 - address 0xE002 C010) bit description (LPC2377/78 and LPC2388)

| PINSEL4 | Pin name | Function when 00 | Function when 01 | Function when 10 | Function when 11 | Reset value |
|-------------|----------|------------------|-----------------------------|------------------|--------------------------|-------------|
| 1:0 | P2.0 | GPIO Port 2.0 | PWM1.1 | TXD1 | TRACECLK ^[1] | 00 |
| 3:2 | P2.1 | GPIO Port 2.1 | PWM1.2 | RXD1 | PIPESTAT0 ^[1] | 00 |
| 5:4 | P2.2 | GPIO Port 2.2 | PWM1.3 | CTS1 | PIPESTAT1 ^[1] | 00 |
| 7:6 | P2.3 | GPIO Port 2.3 | PWM1.4 | DCD1 | PIPESTAT2 ^[1] | 00 |
| 9:8 | P2.4 | GPIO Port 2.4 | PWM1.5 | DSR1 | TRACESYNC ^[1] | 00 |
| 11:10 | P2.5 | GPIO Port 2.5 | PWM1.6 | DTR1 | TRACEPKT0 ^[1] | 00 |
| 13:12 | P2.6 | GPIO Port 2.6 | PCAP1.0 | RI1 | TRACEPKT1 ^[1] | 00 |
| 15:14 | P2.7 | GPIO Port 2.7 | RD2 ^[2] | RTS1 | TRACEPKT2 ^[1] | 00 |
| 17:16 | P2.8 | GPIO Port 2.8 | TD2 ^[2] | TXD2 | TRACEPKT3 ^[1] | 00 |
| 19:18 | P2.9 | GPIO Port 2.9 | USB_CONNECT1 ^[2] | RXD2 | EXTIN0 ^[1] | 00 |
| 21:20 | P2.10 | GPIO Port 2.10 | EINT0 | Reserved | Reserved | 00 |
| 23:22 | P2.11 | GPIO Port 2.11 | EINT1 | MCIDAT1 | I2STX_CLK | 00 |
| 26 = 2 * 13 | P2.12 | GPIO Port 2.12 | EINT2 | MCIDAT2 | I2STX_WS | 00 |
| 27:26 | P2.13 | GPIO Port 2.13 | EINT3 | MCIDAT3 | I2STX_SDA | 00 |
| 29:28 | P2.14 | Reserved | Reserved | Reserved | Reserved | 00 |
| 31:30 | P2.15 | Reserved | Reserved | Reserved | Reserved | 00 |

masque en ou : $(1 \text{ ou } x)=1$ PINSELn |= masque; //force à 1 tous les bits à 1 de masque
 masque en et : $(0 \text{ et } x)=0$ PINSELn &= masque; //force à 0 tous les bits à 0 de masque
 les opérateurs utiles en C : <<y décalage de y bits equivaut à faire * (2^y)
 le dernier opérateur utile : opérateur ~ qui permet une inversion bit à bit

ON veut effacer les bits 26 et 27 du registre PINSEL4 sans toucher aux autres....

étape 1 fabriquer le masque qui selectionne les deux bits :

```
masque = 0x03 << 26;  
//on veut que ce masque soit un masque de forçage à zéro : tous les bits à un  
sauf les deux bits  
masque = ~masque ; // permet une inversion bit à bit : 0 devient 1, et 1 devient 0
```

un effacement se fait par un masque en &

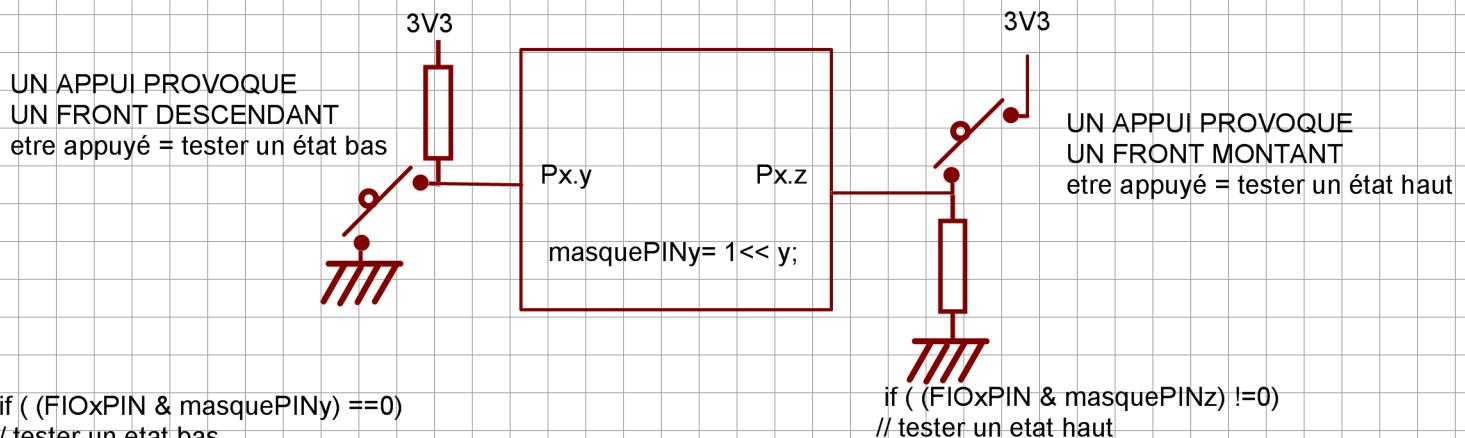
```
PINSEL4&=masque; // directement sinon : PINSEL4 &= ~(0x03 << 26);  
//ou encore en direct : PINSEL4 &= 0xF9FFFFFF;
```

bit 27 26 25 24 : 1001 en binaire = 9

pour définir les pattes en sortie : P4.0 à P4.7

FIO4DIR = 0x000000FF;

Comment est cablé le bouton poussoir ?



tester l'appui nécessite de connaître l'état précédent : un appui est un changement d'état => connaître état précédent

```
etatprecedent = etat;  
etat = FIOxPIN & masquePINy;
```

```
//test front montant : etatprecedent est 0 et etat !=0 // if ((0==etatprecedent) && (etat!=0))  
//test front descendant : etatprecedent !=0 et etat est 0 // if ((etatprecedent != 0) && (0==etat))
```

modification du code : allumé si bouton enfoncé , éteint sinon

```
int main(void)  
{  
init_proc();  
while(1)  
{  
if((FIO2PIN & (1<<13)) == 0 ) //situation où l'appui de bouton est un niveau 0  
{ FIO4PIN = 0xFF;  
/delay_mano(1000000); // par rapport à la persistance rétinienne  
}  
else  
{  
FIO4PIN = 0x00;  
/delay_mano(1000000); // par rapport à la persistance rétinienne  
}  
}
```

chenillard dont la direction dépend d'un bouton

```

int main(void)
{
init_proc();
FIO4PIN= 1<<4;
while(1)
{
if((FIO2PIN & (1<<13)) == 0 ) //situation où l'appui de bouton est un niveau 0
{ if(position <128) {position = position <<1;}
//delay_mano(1000000); // par rapport à la persistance rétinienne
}
else
{
if(position >1) {position = position >>1;}
//delay_mano(1000000); // par rapport à la persistance rétinienne
}
FIO4PIN= position;
}
}

```

UTILISATION d'un périphérique dédié pour détecter les évènements état ou FRONTS

si un évènement dure très peu de temps, on a un risque de le rater par scrutation
on peut déléguer cela à un périphérique dédié

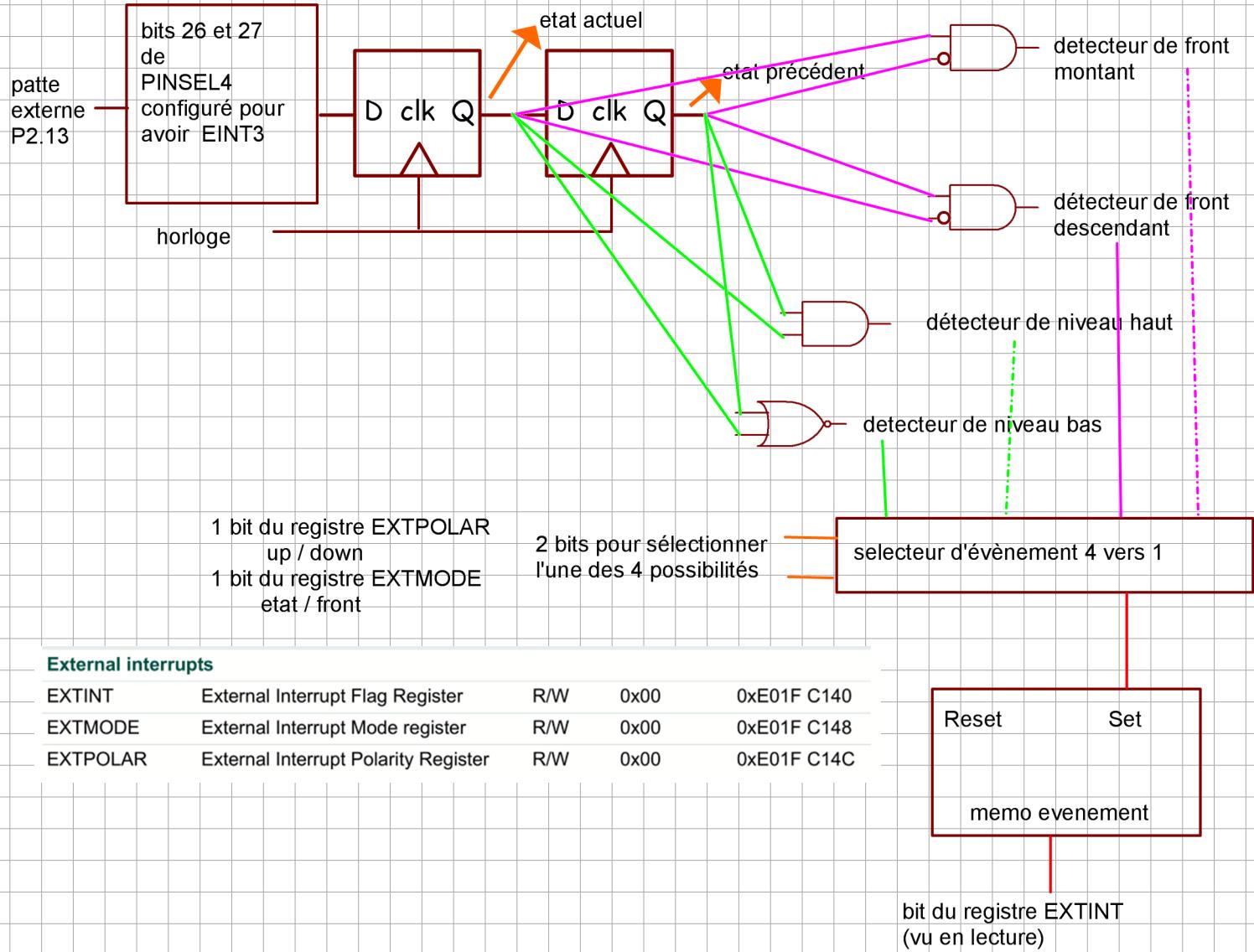


Table 20. External Interrupt Mode register (EXTMODE - address 0xE01F C148) bit description

| Bit | Symbol | Value | Description | Reset value |
|-----|----------|-------|--|-------------|
| 0 | EXTMODE0 | 0 | Level-sensitivity is selected for <u>EINT0</u> . | 0 |
| | | 1 | <u>EINT0</u> is edge sensitive. | |
| 1 | EXTMODE1 | 0 | Level-sensitivity is selected for <u>EINT1</u> . | 0 |
| | | 1 | <u>EINT1</u> is edge sensitive. | |
| 2 | EXTMODE2 | 0 | Level-sensitivity is selected for <u>EINT2</u> . | 0 |
| | | 1 | <u>EINT2</u> is edge sensitive. | |
| 3 | EXTMODE3 | 0 | Level-sensitivity is selected for <u>EINT3</u> . | 0 |
| | | 1 | <u>EINT3</u> is edge sensitive. | |
| 7:4 | - | - | Reserved, user software should not write ones to reserved bits. The value read from a reserved bit is not defined. | NA |

Table 21. External Interrupt Polarity register (EXTPOLAR - address 0xE01F C14C) bit description

| Bit | Symbol | Value | Description | Reset value |
|-----|-----------|-------|---|-------------|
| 0 | EXTPOLAR0 | 0 | <u>EINT0</u> is low-active or falling-edge sensitive (depending on EXTMODE0). | 0 |
| | | 1 | <u>EINT0</u> is high-active or rising-edge sensitive (depending on EXTMODE0). | |
| 1 | EXTPOLAR1 | 0 | <u>EINT1</u> is low-active or falling-edge sensitive (depending on EXTMODE1). | 0 |
| | | 1 | <u>EINT1</u> is high-active or rising-edge sensitive (depending on EXTMODE1). | |
| 2 | EXTPOLAR2 | 0 | <u>EINT2</u> is low-active or falling-edge sensitive (depending on EXTMODE2). | 0 |
| | | 1 | <u>EINT2</u> is high-active or rising-edge sensitive (depending on EXTMODE2). | |
| 3 | EXTPOLAR3 | 0 | <u>EINT3</u> | |

Pas oublier de configurer le PINSEL4 avec les paires de bits 20/21 22/23 24/25 26/27 pour EINT0, EINT1 ... EINT3

Table 113. Pin function select register 4 (PINSEL4 - address 0xE002 C010) bit description (LPC2377/78 and LPC2388)

| PINSEL4 | Pin name | Function when 00 | Function when 01 | Function when 10 | Function when 11 | Reset value |
|---------|----------|------------------|-----------------------------|------------------|--------------------------|-------------|
| 1:0 | P2.0 | GPIO Port 2.0 | PWM1.1 | TXD1 | TRACECLK ^[1] | 00 |
| 3:2 | P2.1 | GPIO Port 2.1 | PWM1.2 | RXD1 | PIPESTAT0 ^[1] | 00 |
| 5:4 | P2.2 | GPIO Port 2.2 | PWM1.3 | CTS1 | PIPESTAT1 ^[1] | 00 |
| 7:6 | P2.3 | GPIO Port 2.3 | PWM1.4 | DCD1 | PIPESTAT2 ^[1] | 00 |
| 9:8 | P2.4 | GPIO Port 2.4 | PWM1.5 | DSR1 | TRACESYNC ^[1] | 00 |
| 11:10 | P2.5 | GPIO Port 2.5 | PWM1.6 | DTR1 | TRACEPKT0 ^[1] | 00 |
| 13:12 | P2.6 | GPIO Port 2.6 | PCAP1.0 | RI1 | TRACEPKT1 ^[1] | 00 |
| 15:14 | P2.7 | GPIO Port 2.7 | RD2 ^[2] | RTS1 | TRACEPKT2 ^[1] | 00 |
| 17:16 | P2.8 | GPIO Port 2.8 | TD2 ^[2] | TXD2 | TRACEPKT3 ^[1] | 00 |
| 19:18 | P2.9 | GPIO Port 2.9 | USB CONNECT1 ^[2] | RXD2 | EXTIN0 ^[1] | 00 |
| 21:20 | P2.10 | GPIO Port 2.10 | <u>EINT0</u> | Reserved | Reserved | 00 |
| 23:22 | P2.11 | GPIO Port 2.11 | <u>EINT1</u> | MCIDAT1 | I2STX_CLK | 00 |
| 25:24 | P2.12 | GPIO Port 2.12 | <u>EINT2</u> | MCIDAT2 | I2STX_WS | 00 |
| 27:26 | P2.13 | GPIO Port 2.13 | <u>EINT3</u> | MCIDAT3 | I2STX_SDA | 00 |
| 29:28 | P2.14 | Reserved | Reserved | Reserved | Reserved | 00 |
| 31:30 | P2.15 | Reserved | Reserved | Reserved | Reserved | 00 |

Partie 2 : Interruptions externes

évenement externe ...

1. Ecrire un premier programme simple qui permet d'allumer ou d'éteindre les LEDs à chaque appui du bouton poussoir. Gérer le bouton poussoir par interruption et non plus par scrutation. N'oubliez pas de modifier la fonction init_GPIO() pour configurer la bonne fonction de la PIN P2.13.

d'évènement !!!!

```
void init_GPIO(void)
{ PINSEL8 = 0;
PINSEL4=0x01 <<26; // pour avoir EINT3
FIO4DIR = 0x000000FF;
EXTMODE = 1<<3; // sensibilité au front
EXTPOLAR = 1<<3; //choix front montant
}
void init_proc(void)
{ init_GPIO(); }
int main(void)
{
init_proc();
FIO4PIN= 0;
while(1)
{ if(EXTINT & (1<<3)) //situation où l'appui de bouton est un niveau 0
{ FIO4PIN ^= 0xFF;
//delay_mano(1000000); // par rapport à la persistance rétinienne
}
}
}
```

on ne regarde plus FIO2PIN& (1<<13)
mais le bit du registre EXTINT qui
mémorise l'évènement !!!!

Lors du test de ce code , on constate qu'on ne rentre pas dans le if tant qu'on ne cree pas un front montant sur la patte P2.3

MAIS à partir de cet évènement , on rentre tout le temps dans le IF ...

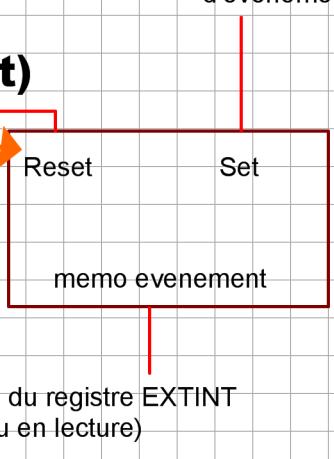
il faut acquitter l'évènement pour dire qu'on a géré cet évènement ! Il faut faire un reset sur la bascule R S :

**Cela se fait en écrivant un 1 (Reset et non /Reset)
sur le bit de EXTINT**

```
int main(void)
{ init_proc();
FIO4PIN= 0;
while(1)
{ if(EXTINT & (1<<3)) //situation où l'appui de bouton est un niveau 0
{ FIO4PIN ^= 0xFF;
EXTINT = 1 << 3; // va provoquer l'effacement du bit
//delay_mano(1000000); // par rapport à la persistance rétinienne
}
}
}
```

bit de EXTINT en écriture

selecteur d'évènement



Partie 2 : Interruptions externes

1. Ecrire un premier programme simple qui permet d'allumer ou d'éteindre les LEDs à chaque appui du bouton poussoir. Gérer le bouton poussoir par interruption et non plus par scrutation. N'oubliez pas de modifier la fonction init_GPIO() pour configurer la bonne fonction de la PIN P2.13.

Pour des raisons d'urgence, on souhaite maintenant réellement avoir une interruption :

le code en cours va être interrompu pour gérer une urgence

On rajoute des modifications de code pour l'interruption :

Oon va dire au gestionnaire d'interruption que l'évènement choisi sur EINT3 doit déclencher une interruption située à une certaine adresse de code :

- configurer l'évènement (déjà vu au dessus)
- acquitter les évènements du passé : EXTINT = 0xFF;
- déclarer où le processeur devra se brancher
- autoriser l'évènement choisi à déclencher une IT

```
void init_EINT3_IT(void)
{ PINSEL4=0x01 <<26; // pour avoir EINT3
EXTMODE = 1<<3; // sensibilité au front
EXTPOLAR = 1<<3; //choix front montant
EXTINT = 0x0F; //effacement général des évènement EINT0 à EINT3
VICVectAddr17 = (unsigned long) IT_EXT3 ;
VICIntEnable = 1<<17; // autorise l'interruption
}
```

Il faut écrire une routine d'interruption pour gérer l'évènement maintenant : NE PAS OUBLIER __irq

```
void IT_EXT3 (void) __irq
{FIO4PIN^=0xFF;
EXTINT = 1<<3 ; //acquittement du périphérique
VICVectAddr=0 ; //acquittement du gestionnaire d'interruption
}
```

indique au compilateur qu'il faut faire une sauvegarde et une restitution de contexte

Ce code va être exécuté immédiatement, en interrompant le fil d'exécution de la fonction main... en conséquence de quoi le test du bit de EXTINT dans le main ne sera plus jamais vrai :

```
int main(void)
{
init_proc();
FIO4PIN= 0;FIO2PIN=0;
while(1)
{ if(EXTINT & (1<<3 ))
//ce test ne sera plus jamais vrai car on aura effacé EXTINT en IT
{ FIO2PIN++;
//delay_mano(1000000); // par rapport à la persistance rétinienne
}
}
```